The Otherside
Web-based Collaborative Multimedia System

Ilias Anagnostopoulos
University of Sheffield Sound Studios
34 Leavygreave Road
S3 7RD, Sheffield
United Kingdom
I.Anagnostopoulos@sheffield.ac.uk

Abstract

The Otherside is an advanced open-source multimedia system, designed to run as a server application. It is essentially a sound synthesis system that allows for the creative collaboration of a number of people or machines over computer networks. Unlike most similar attempts to create network-based audio applications, the Otherside does not require any specialized audio software or massive amounts of computer processing power. The audio processing is done on the server side and directed to listeners using a simple Internet Radio protocol. Any listener can then choose to participate in the sound creation process by using a simple web-browser-based chatroom, or by utilising their own advanced systems for network control.

Keywords

Networks, Collaboration, Server, Streaming, OSC

1 Introduction

1.1 Creative Collaboration

The Otherside Server provides a platform for the creative collaboration of any number of people, regardless of their understanding of music, computer programming or networking. It is platform-independent, meaning that it will run properly on any operating system and any computer architecture. This makes it easily accessible to a large number of people who can work together and even communicate, during the course of their session with Otherside.

People can create music together, from randomly changing values and creating random sonic events to carefully crafting their own software that will interface with the Otherside like an additional musical instrument. It is an entrance to the world of computer music for people who have no such previous experience. It can also serve as an advanced experimentation platform for users who are already familiar with the protocols and principles involved. It is open and accessible to several different protocols and adheres to the open-source mentality, meaning that users can get involved to the extent of becoming developers of the Otherside system.

1.1.1 Computer Network Collaboration

Computer Networks are nowadays dominating the world through the internet. However, most of the well-established network applications are very simple and are not particularly demanding in processing power and network bandwidth. Digital Signal Processing is a complicated task that only recently became a "household" application on personal computers. Careful planning and consideration is required to overcome some of the difficulties posed by computer networks and their use by artistic collaboration systems.

One of these problems is that network connections and especially the internet often involve extremely long distances between two users, thus inevitably data can not be transferred seamlessly. There is an evident latency issue that is being treated differently by such project developers. Jeorg Stelkens discusses this in a paper about his own network collaboration system[1], proposing the integration of latency in the creative process, as a unique characteristic of the computer network as a medium. The Otherside does not attempt to overcome or to cover-up latency, unlike most projects mentioned by Stelkens. It merely accepts latency as a fact and uses the amount of users simultaneously transmitting data to create a sonic situation where events can either be left to chance, creating aleatory music[2], or forcing a user to take the amount of latency into consideration when crafting a sonic event in order to be able to predict the outcome. This approach is based on the principle of integrating the latency in the creative process and passing on its acceptance as a fact to the users, who will have to take it into account when using this system.

1.2 Related Creative Development

1.2.1 NetPD

An inspiring project, leading to the initial thoughts for creating the Otherside was NetPD[3]. NetPD is based on Pure-Data[4] and enables a group of people to collaborate over a computer network. The idea of collaboration is common between NetPD and the Otherside, but the Otherside attempts to break the limit of keeping it only between users of Pure-Data, who would have to be comfortable with data-flow programming and networking. The Otherside allows users to become involved as much as they please. NetPD also features a chat facility within Pure-Data, for communication between participants. Otherside is based on IRC, a well-known chat interface, but goes much further than just using it for communication. It is worth noting that it is technically possible to connect NetPD and the Otherside and create a collaboration between the two projects.

1.2.2 Pure-Data and Data-Flow Programming

Pure-Data was created by Miller Puckette, as an open-source project, similar to his Max[5] program. It is a data-flow programming language geared towards audio/visual output and control. Pure-Data is a direct descendant of Max, written in 1988[6] by Miller Puckette while he was in the Institut de Recherche et Coordination Acoustique/Musique, Paris, France. David Zicarelli together with Puckette later wrote MSP as an addition to Max, thus creating Max/MSP, enabling audio-rate digital singal processing. Max was only capable of control rate processing, as computers in 1988 were not fast enough to handle audio-rate signal processing. Nowadays data-flow programming is used widely for audio applications, as it is a very powerful and versatile way of interfacing with various control protocols. Computers nowadays have enough processing power to support heavy digital signal processing applications and networking speeds are high enough to be used creatively in interactive applications. This makes data-flow programming languages an interesting option for users who want to explore the field of complex control in sound generation systems.

2 Technical Description

2.1 Operating System

The Otherside is based on open-source principles, being built on the Ubuntu Linux 8.04 Server Edition operating system. The operating system was chosen as it has proved to be very stable as a server system while being directly compatible with all the audio software and techniques that were to be used. It is also an operating system that is very open to customization, something that was necessary to ensure high performance under the load of audio processing and server applications.

The preparation of the computer system on which the Otherside was tested started with the installation of the base Ubuntu 8.04 Server system, replacing the Linux Server Kernel with the Linux 2.6.24 RT[1] Kernel, which allows audio processing to be done with Real-Time priority. Since audio processing is one of the main functions of the Otherside, real-time implementation was an important step towards high performance. To enable the audio to function on the Ubuntu Server operating system the installation of ALSA[7] was also required.

2.2 Web interface

To achieve an extreme level of accessibility by users who have no interest in turning their personal computer into an audio workstation, all functions of the Otherside are accessible from a simple website interface.

The starting point is a website that is powered by the Apache 2 HTTP server[8], the most widely used HTTP server on the internet. This links to a Streaming Audio Server, an IRC Chat Interface, a help document simply explaining the basics of the Otherside to the users and an email link for communication with the administrator of the system. Apache 2 was chosen due to the fact that all other HTTP servers under investigation[2] were simply not as advanced and reliable, something which is also evident by the widespread use of the Apache on the internet.

The Streaming Audio Server is using the Icecast 2 Server[9] to provide a compressed audio stream produced by the Otherside to an audience with an internet connection and a media player on their computer. The Icecast 2

[1] An alternative Kernel, with Real-Time capabilities

[2] Examples of other HTTP Servers that were considered instead of Apache include Mathopd, micro-httpd, mini-httpd and nanoweb

Server receives a compressed audio stream, utilising MPEG Layer 3 compression, forwarding it to a number of listeners over a computer network. This is based on the established Internet Radio Station practice of using an encoder that streams audio to a Streaming Audio Server to be forwarded to any number of listeners. The difference here is that instead of encoding sound files, the encoder is directly encoding and streaming the computer-generated sound of the synthesis engine that powers the Otherside. The link from the main website redirects a user to the website interface of the Icecast 2 Server, which contains details about the audio stream and a link for listening. Icecast 2 was chosen because it is a very well-made open-source Streaming Server with a friendly user interface.

The control interface was a challenge as there were too many functions that needed to be implemented in a simple and accessible interface. The choice of an IRC Chat interface as the control interface was made as the nature of IRC already implemented a lot of the functions that were needed, without the need of reinventing the wheel. Several different alternatives for the IRC Server software were tried, but most of the common servers[3] proved to be unsuitable for the kind of use that was about to be adopted. The RFC1459[10] and RFC2813[11] specifications include several functions geared towards huge IRC Server networks, which were a burden on CPU power and configuration time, therefore clearly not wanted in the Otherside. The server software which was finally used was the IRC-Net IRC 2 Server, as it was very simple to configure, reliable and somewhat modular in design, allowing the administrator to engage or disengage any modules needed for the task. By default, it comes with the bare minimum necessary to function, which is exactly what the Otherside requires.

Usually, an IRC Network has ways of registering nick-names and chatrooms. This is to allow frequent users to always use their own nickname while prohibiting anybody else from using it, while there is also a service for leaving messages to registered users that are currently offline. These are collectively known as IRC Services. The decision against using any IRC Services Provider package came naturally at this stage, as the IRC Server used with the Otherside is merely a control interface, therefore there is no need for registering a nick-name or a chat-room.

An IRC Server also has the ability to connect to other IRC Servers, forming IRC Networks. This was one of the most impressive features of the IRC protocol, as it means that there is room for a great amount of evolution for the Otherside, by connecting to other Othersides and forming a huge Otherside Network in a very simple and efficient way. This is also an easy way to get extensive load off of one server's central processing unit, by interfacing several computers to run the Otherside System and sharing the load of processing in a clustered-computing fashion.

However, an IRC Server is only the back-end, requiring a Client program[4] for user interaction. Most Client programs assume a certain level of understanding from the user about the IRC Protocol. Investigation on this matter led to the use of CGI:IRC, a very smart way of creating a web-browser interface for the IRC Server, thus eliminating the requirement for an IRC Client program to be installed on the client computer. CGI:IRC runs on the server machine and creates a CGI-based web-page that can run on the majority of web-browsers, allowing the user to connect to and use the IRC Server from this web-based client.

This IRC interface allows a number of users to connect to a chat-room and type in messages. The interaction between the users and the server was achieved by using an IRC Infobot, a computer program that sits on an IRC Server, posing as a human user, that replies to certain messages with predefined actions. These messages usually contain information regarding the Server. This idea let to the creation of an Infobot-like program that would listen for specific messages, but instead of replying back to the user that sent the message, it would respond by directing them to the Sound Synthesis Engine. After some investigation, it became evident that there was nothing similar readily available. Thus came the decision to create an original control-bot using a computer programming language. Consideration of C/C++[12] and Python[13] led to a decision against C/C++ as it was lacking some of the easy-to-use functions of Python for string formatting. By studying some exist-

[3]Servers that were considered include the Undernet IRC Daemon, the Ratbox IRC Daemon and the Hybrid IRC Daemon

[4]Common IRC client programs include mIRC for the Windows operating system and bitchX for POSIX systems

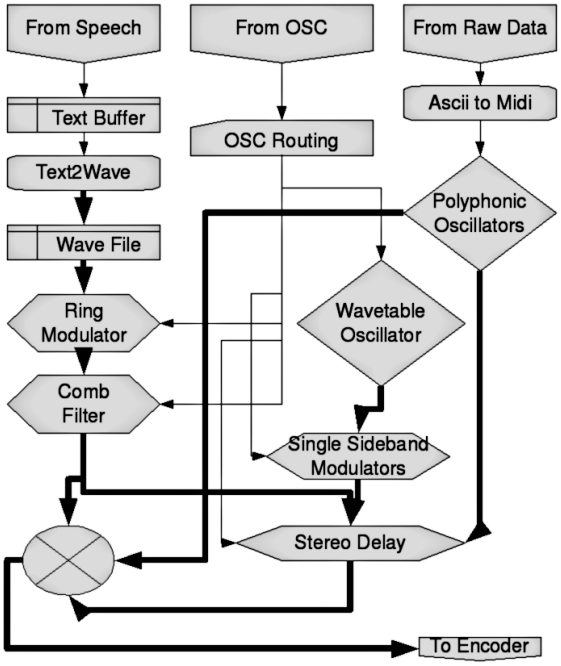
ing Infobots such as the XoR bot skeleton[14] and the Redland Bot[15] it became clear how they worked and ideas were implemented in the new code. The source code was loosely based on the XoR bot skeleton as it had some interesting string formatting examples. The SimpleOSC Library[16] for the Python programming language enabled the reformatting and redirection of messages to the Sound Synthesis Engine, using the extremely versatile OSC protoco[17]. The program also uses the Sockets Library for TCP/UDP network connections to connect to the IRC Server and to redirect all undefined messages sent by users to a module on the Sound Synthesis Engine that treats it as raw American Standard Code for Information Interchange data. Finally, some of the existing Infobot funtions of the XoRbot were retained almost intact, especially regarding the IRC Protocol channel functions and also to include a short help message and the basis for future Musical Instrument Digital Interface implementation.

The users that log on to the chat-room see a user with the nick-name "Otherbot", which is the bot that interfaces the IRC Server with the Sound Synthesis Engine. The only thing they need to do to control the Sound Synthesis Engine is send messages to the bot as if they were chatting to a friend, from their web-browser window.

2.3 Sound Synthesis Engine

The Sound Synthesis Engine is based on a version of the Pure-Data data-flow programming software, called PD-extended. It is modular in design and consists of a Polyphonic Additive Synthesis[18] Module, a Wavetable Synthesis[19] Module, a Single Sideband Modulation[20] Module, a Ring Modulation[21] Module, a Comb Filter[22] Module, a Stereo Delay Line[23] Module and a Speech Synthesis Module. The Speech Synthesis Module is not based on PD-extended. Instead, it is based on the Festival Speech Synthesis System[24], developed by the Centre for Speech Technology Research of the University of Edinburgh, UK. The particular design of the sound synthesis engine was geared more towards the demonstration of the capabilities of the control interface, rather than a complete creative system. There is much more that can be done with the synthesis system and the administration setup of the Otherside makes it easy to update the existing patches or upload new ones, adding to the system.

Figure 1: Block Diagram



The Otherbot outputs three kinds of messages towards the Speech Synthesis Engine. The easiest to explain is probably the Raw ASCII Data. When a user types a message in the chatroom that is not recognised as a predefined command, the bot establishes a User Datagram Protocol connection to port 9998 of the server and redirects the full message. UDP Port 9998 is opened by a PD-extended object listening for inbound connections, thus acting as a daemon program. The data is received as ASCII integer values. These are treated as MIDI pitch values and are directed to a polyphonic additive synthesis module, which creates the equivalent of each MIDI message received, spatially panning it according to the number of characters received in a message. The odd numbers are panned to the left while the even ones are panned to the right. Therefore a message length of three characters would have the first and third characters panned to the left while the second character is panned to the right, after being converted from characters to ASCII values representing MIDI pitch values and then synthe-

sized to sound. The sound is then directed to a Stereo Delay Module for further processing. The spatialization in this module is evidently very simple, something which will be rectified in future versions, by replacing the current algorithm with one that does proportional panning. It currently acts as a mere demonstration of the potential of the Otherside.

When a message typed in the chatroom matches a predefined command recognised by the Otherbot, this is redirected as OSC data by establishing a UDP connection to port 9997 of the server. UDP Port 9997 is opened by a PD-extended object designed to receive OSC data and direct it to the equivalent OSC path. These OSC paths direct data to the appropriate objects in the PD-extended patch, where they control certain functions, such as the speed at which the wavetable is read or the delay time and feedback.

The Wavetable Synthesis Module consists of two saved wavetables. One is used to define a waveform while the other is used to define the speed at which the waveform is being reproduced. When the speed is altered using the appropriate OSC command, the waveforms are being read at different speeds, so the whole process can be sped up or slowed down according to taste.

The outputs of the Wavetable Synthesis Module are directed to two Single Sideband Modulators. These are used primarily for frequency shifting and general "warbling" effects. The output is then directed to the Stereo Delay Module.

The Speech Synthesis module is a bit more complicated since it does not solely rely on PD-extended to complete its task. If a message in the chatroom is preceded by the "talk" command, then anything following the command is being written in a buffer text file. Once this is done, the bot calls a system function that runs Text2Wave, a program that comes with the Festival Speech Synthesis System that converts text files to audio files. The command tells Text2Wave to use the buffer text file as an input file and synthesize the contents of this file to sound, saving it as a PCM Wave file. The PD-extended module that deals with the Speech Synthesis implementation looks for this file and plays it in a loop until the buffer text file is altered. When this happens, the PCM Wave file is altered as well, so the next time the loop is started it starts playing the new file. The Speech Synthesis module sound output is then fed to a Ring Modulator. The Ring Modulator module multiplies the sound with a carrier wave. The frequency of the carrier wave is controllable by OSC, enabling a user to actively control how the voice is effected during playback.

The output of the Ring Modulator module is directed to the Comb Filter module. This creates four instances of the input signal and combines them, applying slight delays to each instance. The level of each of the four instances is changed sequentially according to a time variable. The time variable can be altered via OSC as well. The output of the filter is directed to the Stereo Delay Module.

The Stereo Delay Module is a delay line with a feedback loop. It was based on an earlier design of the author that also had a Graphical User Interface. This was used on the "Weird Synthesizer for Weird People"[25] project and was based on a design by Jason Plumb[26].

2.4 Encoding

The master signal output of the PD-extended patch is not directed to the Digital/Analog Converter object as is most often the case, since we are not interested in directing the sound to the sound-card of the computer it runs on. Instead, it is sent directly to the encoder, an "mp3cast" object that is part of the "Unauthorized" library[27], written by Yves Degoyon. This encodes the sound to the desired format. In this case, it uses the "LAME" library[28] to encode the audio into an MPEG Layer 3 stream and send it to the Icecast 2 Server, directly from PD-extended.

Several different ways of doing the conversion and streaming to the server had been investigated, prior to deciding on the final solution. Methods that were investigated included compiling Darkice[5] with LAME library support to enable it to encode using the MPEG Layer 3 format. However, Darkice required an input by a program that was compatible with it. An early experiment was to use a "DAC" object in PD-extended, together with Jack[6], a signal routing program for ALSA. Darkice had to be recompiled with support for Jack. The output of the "DAC" object was then disconnected from the sound-card input and was directed to the input of Darkice instead. However, this set-up re-

[5] An audio streamer, written by Akos Maroy

[6] Jack is a virtual patchbay for signal routing

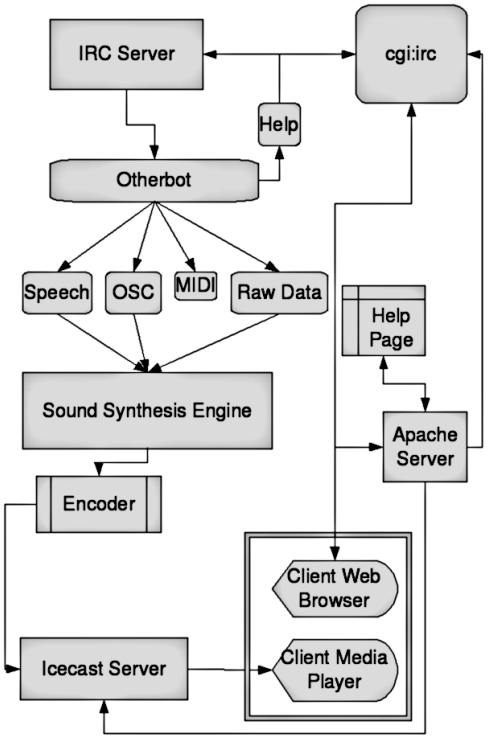
quired two additional programs to be compiled, installed and running on the server, Darkice and Jack Daemon. Although this might have been more efficient used with certain computer systems, in this case it proved to be a burden on CPU power.

2.5 Creating the Bridge

So far, it has been made clear that the Otherside depends on several existing programs, with a bridging system that makes it all work together towards the desired result. The IRC interface acts as the main bridging point. The users send messages to a bot and this bot sends commands to the rest of the programs. This works by listening for specific key phrases. If it detects any of the key phrases in the beginning of a message sent by a user, it interprets the text that follows in the appropriate manner. These key phrases are help, osc, talk, and midi. If the user types "help" in the chatroom or in a private message to the bot, they will get a help message in response. If they type "midi", they will get an information message in response, letting them know that midi has not been implemented yet. "Talk" sends the text following the command to the Festival Speech Synthesis System that converts this into a Wave file. OSC control is slightly more complicated. The user needs to type in the arguments needed by the receiving end, which is PD. The message starts with "osc" as in the previous examples, this phrase is followed by the OSC path, for example `/reverb/time`, and the values that we want to pass to this path, for example `800`. The whole message would be `osc /reverb/time 800`. This tells the bot to interpret `/reverb/time 800` as an OSC message, forwarding it to PD. PD receives this message on UDP port 9997, and redirects the value, 800, to the appropriate path. In effect, this sets the reverb time to 0,8 seconds.

This way of parsing OSC messages might at first seem counter-intuitive. However, considering that we are doing this over IRC, a few traditional possibilites immediately come to mind. For instance, a user can set up their own IRC bot, connect it to that chatroom and instruct it to receive MIDI messages from a MIDI controller connected to the user's computer, convert them to OSC, and send them through a private chat to the OtherBot in the manner specified above. This would create a MIDI-to-OSC bridge, over a network. While manually typing an OSC message is probably not of much use in itself, it can open up amazing possibilities for automated control ideas to be implemented.

Figure 2: General Block Diagram



On the other hand, there will probably be a fair amount of human communication taking place in any IRC chatroom. This data can be used as a kind of control in itself. All messages that the bot receives, but cannot interpret as any known message type, are stripped of the IRC commands that come with them and are sent to PD as raw text messages. PD converts it to ASCII data and uses it. So even when the bot receives no evidently useful data, it still "recycles" seemingly random data to keep things in motion.

3 Conclusion

The Otherside is unique in several ways, compared to other examples of network collaboration systems. First and foremost, from a technical point of view, all processing is done on the server side and the sound is transfered to the listeners and users as an audio stream. This means that all users are listening to the same sonic out-

put. A common problem associated with computer network collaboration systems is the difficulty is synchronisation of the audio due to network latency, especially in a situation where the collaboration might be between several users in the same room. If each of the users is producing audio independently, it is technically impossible to ever synchronize the output. However, in the Otherside system, since all users are working towards a common sonic piece that is being created on the server side, only one computer needs to act as the audio reproduction client in each such setting. Therefore, in the collaboration situation where users are in the same room, the audio only needs to be streamed to one computer, coming out of one pair of loudspeakers. There are no synchronization problems, since all the users are doing is transmitting control data to the server, which creates one sonic artwork by synthesizing transmitted data in the order of arrival.

In the case of a network of Othersides, the servers all share the same user control data as IRC servers share all data between them when networked. However, each Otherside Server runs its own sound synthesis engine so audio data does not need to be transmitted between servers. All servers will create the exact same version of the audio stream, since they have received the same control data, and stream it independently. This is extremely useful as users can stream the audio off their local server, while being able to perceive the input of a user connected to a different Otherside Server. This function makes the possibility for a true global collaboration very realistic.

The use of IRC as the main bonding point that controls everything allows for simple and effective automation methods to be used. Code from IRC bots and utilities can be re-used and altered, thus making it very easy for a programmer to get started in creating a control interface to work with the Otherside.

There is also an evident social side to the Otherside system, since the use of an IRC protocol allows users to easily communicate as they would in normal chat-rooms. They are given the opportunity to start their own chat-rooms and form "teams" of users that cooperate for a desired sonic experiment. This can easily grow into a network of Otherside Servers in different parts of the world, who could collaborate in creating sound or developing the Otherside system. The IRC protocol is currently not being used at its full potential for chat services. It can easily evolve into a full IRC chat interface, with services enabling users to have a personalized identifier/nickname and build permanent subject-specific chatrooms. The Otherside presents the world with the opportunity to turn the idea of computer network collaboration platforms into a global social network.

Location

The Otherside server can be accessed during the Linux Audio Conference 2009 on the following URL: `http://otherside.servebeer.com/otherside`

References

[1] Joerg Stelkens. peersynth: A p2p multiuser software synthesizer with new techniques for integrating latency in real time collaboration. In *Proceedings of the 2003 ICMC*. Singapore: ICMA, 2003.

[2] Encyclopaedia Britannica. Aleatory music, 2008. http://www.britannica.com/EBchecked/topic/13676/aleatory-music, Accessed on 02/09/2008.

[3] Roman Haefeli. Netpd, 2008. http://www.netpd.org/, Accessed on 01/09/2008.

[4] Miller Puckette, IOhannes M. Zmoelnig, and the PD Community. Pure-data, 2008. http://puredata.info/, Accessed on 01/09/2008.

[5] Miller Puckette. Cycling 74, 2008. http://www.cycling74.com/products/max5/, Accessed on 01/09/2008.

[6] Max Matthews. *The theory and techniques of electronic music*, chapter Foreword. Singapore: World Scientific, 2007. http://www-crca.ucsd.edu/ msp/techniques/latest/book-html/node5.html, Accessed on 01/09/2008.

[7] Jaroslav Kysela et al. Advanced linux sound architecture, 2008. http://www.alsa-project.org/main/index.php/Main_Page, Accessed on 01/09/2008.

[8] The Apache Software Foundation. Apache 2, 2008. http://httpd.apache.org/, Accessed on 01/09/2008.

[9] The XIPH Open Source Community. Icecast 2 server, 2008. http://www.icecast.org/, Accessed on 01/09/2008.

[10] Jarkko Oikarinen and Darren Reed. *The IRC RFC-1459*, 1993. http://www.irc.org/tech_docs/ircnet/rfc1459.txt, Accessed on 01/09/2008.

[11] Christophe Kalt. *The IRC RFC-2813*, 2000. http://www.irc.org/tech_docs/ircnet/rfc2813.txt, Accessed on 01/09/2008.

[12] Dennis M. Ritchie. The development of the c language. In Thomas J. Bergin, Jr. and Richard G. Gibson, Jr., editors, *History of Programming Languages-II*. New York: ACM Press, 1993. http://cm.bell-labs.com/cm/cs/who/dmr/chist.html, Accessed on 01/09/2008.

[13] Python Software Foundation. Python programming language, 2008. http://www.python.org/, Accessed on 01/09/2008.

[14] ArchGh0ul. Python irc bot skeleton, 2007. http://www.rohitab.com/discuss/index.php?showtopic=24081, Accessed on 01/09/2008.

[15] Dave Beckett. Julie, formerly known as redlandbot, 2008. http://crschmidt.net/julie/, Accessed on 01/09/2008.

[16] Daniel Holth, Clinton McChesney, and the ixi software group. Simpleosc 0.2.5, 2008. http://www.ixi-software.net/content/download/simpleosc0.2.5.zip, Accessed on 01/09/2008.

[17] Matt Wright. The Open Sound Control 1.0 Specification, 2002. http://opensoundcontrol.org/spec-1_0, Accessed on 01/09/2008.

[18] Gordon Reid. Part 14: An introduction to additive synthesis. *Sound on Sound*, June 2000. http://www.soundonsound.com/sos/jun00/articles/synthsec.htm, Accessed on 02/09/2008.

[19] Robert Bristow-Johnson. Wavetable synthesis 101, a fundamental perspective, 2008. http://www.musicdsp.org/files/Wavetable-101.pdf, Accessed on 02/09/2008.

[20] MathWorks Team. Single sideband modulation via the hilbert transform, 2008. http://www.mathworks.com/products/signal/demos.html?file=/products/demos/shipping/signal/hilberttransformdemo.html, Accessed on 02/09/2008.

[21] Scott Lehman. Ring modulation, 2007. http://www.harmony-central.com/Effects/Articles/Ring_Modulation/, Accessed on 02/09/2008.

[22] Julius O. Smith III. *Physical Audio Signal Processing*, chapter Comb Filters. Stanford University, California, May 2008 edition, 2007. http://ccrma.stanford.edu/ jos/pasp04/Comb_Filters.html, Accessed on 02/09/2008.

[23] Julius O. Smith III. *Physical Audio Signal Processing*, chapter Delay Lines. Stanford University, California, May 2008 edition, 2007. http://ccrma.stanford.edu/ jos/pasp/Delay_Lines.html, Accessed on 02/09/2008.

[24] Alan W. Black, Richard Caley, and Paul Taylor. *The Festival Speech Synthesis System*, 1.4, for festival 1.4.1 edition, 1999. http://fife.speech.cs.cmu.edu/festival/manual-1.4.1/festival-1.4.1.ps.gz, Accessed on 02/09/2008.

[25] Ilias Anagnostopoulos. Weird synthesizer for weird people, 2008. http://iforgotthem.tripod.com/ift.htm, Accessed on 01/09/2008.

[26] Jason Plumb. Noisybox, 2007. http://noisybox.net/computers/pd/, Accessed on 01/09/2008.

[27] Yves Degoyon. Unauthorized pure data, 2008. http://ydegoyon.free.fr/software.html, Accessed on 02/09/2008.

[28] Mike Cheng, Mark Taylor, et al. Lame ain't an mp3 encoder, 1988. http://lame.sourceforge.net/index.php, Accessed on 02/09/2008.